Assignment 2: Working with data

Let's work through an example of a time series of average monthly temperatures for four cities from 1950 to the present. Data can be found at https://www.ncdc.noaa.gov/cag/city/time-series/

To get data for a city, select *Average Temperature*, *All Months*, *Start Year*, *End Year*, *State*, and *City*. Time series for several cities can be combined into a single dataset using Python or Excel.

```
from google.colab import drive
drive.mount('/content/drive')

from IPython.display import Image, display

datadir = 'data/'
imagesdir = 'images/'
renderer = 'colab'

datadir = '/content/drive/MyDrive/Colab Notebooks/Ass2/data/'
imagesdir = '/content/drive/MyDrive/Colab Notebooks/Ass2/images/'

def display_images(images, dir=imagesdir):
    for image in images:
        display(Image(dir + image))
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
```

```
!pip install plotly
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly) (9.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly) (24.2)
```

```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go


# read the dataset
filename = datadir + 'cityTempsSeries.csv'
df = pd.read_csv(filename)
df
```

	Date	Cleveland	Miami	NYC	SF
0	195001	35.0	73.6	41.0	44.0
1	195002	27.7	70.5	30.8	49.0
2	195003	31.5	72.2	35.8	50.6
3	195004	41.6	71.4	48.1	54.3
4	195005	59.7	79.5	58.5	55.7
...	...	...	...	...	...
861	202110	61.6	81.2	62.0	62.7
862	202111	41.8	73.1	46.3	58.6
863	202112	40.5	74.5	43.8	50.2
864	202201	23.8	68.7	30.3	52.6
865	202202	30.5	73.5	37.3	54.6

866 rows × 5 columns

The *date* column has format *yyyymm* and there is one column per city. Temperatures are given in Farenheit.

```
df.columns
```

```
Index(['Date', 'Cleveland', 'Miami', 'NYC', 'SF'], dtype='object')
```

What are the attributes?

- *dates*: ordinal
- *cities*: categorical (unordered)
- *temperature values*: numerical

We can extract the columns by name.

```
print(df['Date'])
print(df.Miami)
```

```
0      195001
1      195002
2      195003
3      195004
4      195005
        ...
861    202110
862    202111
863    202112
864    202201
865    202202
Name: Date, Length: 866, dtype: int64
0      73.6
1      70.5
2      72.2
3      71.4
4      79.5
       ...
861    81.2
862    73.1
863    74.5
864    68.7
865    73.5
Name: Miami, Length: 866, dtype: float64
```

Note the *Date* column is of type integer. Let's convert this column to a Python *datetime* object.

```
df['Date'] = pd.to_datetime(df['Date'], format='%Y%m')
df
```

	Date	Cleveland	Miami	NYC	SF
0	1950-01-01	35.0	73.6	41.0	44.0
1	1950-02-01	27.7	70.5	30.8	49.0
2	1950-03-01	31.5	72.2	35.8	50.6
3	1950-04-01	41.6	71.4	48.1	54.3
4	1950-05-01	59.7	79.5	58.5	55.7
...	...	...	...	...	...
861	2021-10-01	61.6	81.2	62.0	62.7
862	2021-11-01	41.8	73.1	46.3	58.6
863	2021-12-01	40.5	74.5	43.8	50.2
864	2022-01-01	23.8	68.7	30.3	52.6
865	2022-02-01	30.5	73.5	37.3	54.6

866 rows × 5 columns

Let's extract the month and year from the datetime object and add these as new columns. We'll represent the year as an integer and the month as a month name abbreviation.

```
d = df['Date'][1]
print(d)
print(d.month)
print(d.year)
```

```
1950-02-01 00:00:00
2
1950
```

```
# get month name abbreviations
import calendar

for i in range(1, 13):
  print(calendar.month_abbr[i])
```

```
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
```

```
# add new columns to frame
df['Year'] = df['Date'].apply(lambda date: date.year)
df['Month'] = df['Date'].apply(lambda date: calendar.month_abbr[date.month])
df
```

	Date	Cleveland	Miami	NYC	SF	Year	Month
0	1950-01-01	35.0	73.6	41.0	44.0	1950	Jan
1	1950-02-01	27.7	70.5	30.8	49.0	1950	Feb
2	1950-03-01	31.5	72.2	35.8	50.6	1950	Mar
3	1950-04-01	41.6	71.4	48.1	54.3	1950	Apr
4	1950-05-01	59.7	79.5	58.5	55.7	1950	May
...	...	...	...	...	...	...	...
861	2021-10-01	61.6	81.2	62.0	62.7	2021	Oct
862	2021-11-01	41.8	73.1	46.3	58.6	2021	Nov
863	2021-12-01	40.5	74.5	43.8	50.2	2021	Dec
864	2022-01-01	23.8	68.7	30.3	52.6	2022	Jan
865	2022-02-01	30.5	73.5	37.3	54.6	2022	Feb

866 rows × 7 columns

Let's reorder the columns and drop the *Date* column.

```
cols = list(df.columns)
cols
```

```
['Date', 'Cleveland', 'Miami', 'NYC', 'SF', 'Year', 'Month']
```

```
newcols = cols[-2:] + cols[1:-2]
newcols
```

```
['Year', 'Month', 'Cleveland', 'Miami', 'NYC', 'SF']
```

```
df = df[newcols]
df
```

	Year	Month	Cleveland	Miami	NYC	SF
0	1950	Jan	35.0	73.6	41.0	44.0
1	1950	Feb	27.7	70.5	30.8	49.0
2	1950	Mar	31.5	72.2	35.8	50.6
3	1950	Apr	41.6	71.4	48.1	54.3
4	1950	May	59.7	79.5	58.5	55.7
...	...	...	...	...	...	...
861	2021	Oct	61.6	81.2	62.0	62.7
862	2021	Nov	41.8	73.1	46.3	58.6
863	2021	Dec	40.5	74.5	43.8	50.2
864	2022	Jan	23.8	68.7	30.3	52.6
865	2022	Feb	30.5	73.5	37.3	54.6

866 rows × 6 columns

What are the attributes?

- *Year*: ordinal
- *Month*: ordinal
- *cities*: categorical (unordered)
- *temperature values*: numerical

Let's make another dataframe with the same data as `df` but a different format. We *unpivot* `df` so that our new dataframe `df2` has a 'city' column with city name. For each record in `df`, there is one record in `df2` for every city. Dataframe `df2` consists of these fields: year, month, city, and temperature. In pandas, we unpivot using `melt`.

```
cities = list(df.columns[2:])
dfu = pd.melt(df,  id_vars=['Year', 'Month'], value_vars=cities, var_name='city', value_name='temp')


print(df.head(5), '\n\n', dfu.head(5))
```

```
   Year Month  Cleveland  Miami   NYC    SF
0  1950   Jan       35.0   73.6  41.0  44.0
1  1950   Feb       27.7   70.5  30.8  49.0
2  1950   Mar       31.5   72.2  35.8  50.6
3  1950   Apr       41.6   71.4  48.1  54.3
4  1950   May       59.7   79.5  58.5  55.7

    Year Month       city  temp
0  1950   Jan  Cleveland  35.0
1  1950   Feb  Cleveland  27.7
2  1950   Mar  Cleveland  31.5
3  1950   Apr  Cleveland  41.6
4  1950   May  Cleveland  59.7
```

```
print(dfu[(dfu.Year == 1950) & (dfu.Month == 'Jan')])
```

```
       Year Month       city  temp
0      1950   Jan  Cleveland  35.0
866    1950   Jan      Miami  73.6
1732   1950   Jan        NYC  41.0
2598   1950   Jan         SF  44.0
```

For the exercises that follow, you may use either `df` or `dfu`, whichever you find more convenient. When you write a function that inputs an *unpivoted* dataframe like `df2`, append 'U' to the function name.

Exercise 1

Go to https://www.ncdc.noaa.gov/cag/city/time-series/ and retrieve time-series temperature data for four to six cities of your choice. To get data for a city, select *Average Temperature*, *All Months*, *Start Year*, *End Year*, *State*, and *City*.

Use Excel to combine these datasets into a single file *cityTempsSeries.csv* with temperature data for your cities. This has the same format as the file read at the top of this notebook, but it has data for your chosen cities.

Then use the techniques used in this notebook above to create a dataframe with the format given above.

```
#Import csv with data
filename = datadir + 'Assignment2.csv'
df = pd.read_csv(filename)
df.head()
```

	Date	Honolulu	Rochester	Hartford	Buffalo
0	195001	72.7	8.8	32.0	33.6
1	195002	73.7	11.1	27.9	28.9
2	195003	73.8	15.1	28.7	28.6
3	195004	73.9	20.6	32.8	31.3
4	195005	74.4	27.7	37.7	36.5

```
df['Date'] = pd.to_datetime(df['Date'], format='%Y%m')
df['Year'] = df['Date'].apply(lambda date: date.year)
df['Month'] = df['Date'].apply(lambda date: calendar.month_abbr[date.month])
cols = list(df.columns)
newcols = cols[-2:] + cols[1:-2]
df = df[newcols]
df
```

	Year	Month	Honolulu	Rochester	Hartford	Buffalo
0	1950	Jan	72.7	8.8	32.0	33.6
1	1950	Feb	73.7	11.1	27.9	28.9
2	1950	Mar	73.8	15.1	28.7	28.6
3	1950	Apr	73.9	20.6	32.8	31.3
4	1950	May	74.4	27.7	37.7	36.5
...	...	...	...	...	...	...
899	2024	Dec	78.2	48.7	53.4	53.0
900	2025	Jan	75.1	15.2	26.0	24.2
901	2025	Feb	75.1	15.7	27.4	25.0
902	2025	Mar	76.0	23.4	32.2	30.1
903	2025	Apr	76.8	29.1	36.9	34.5

904 rows × 6 columns

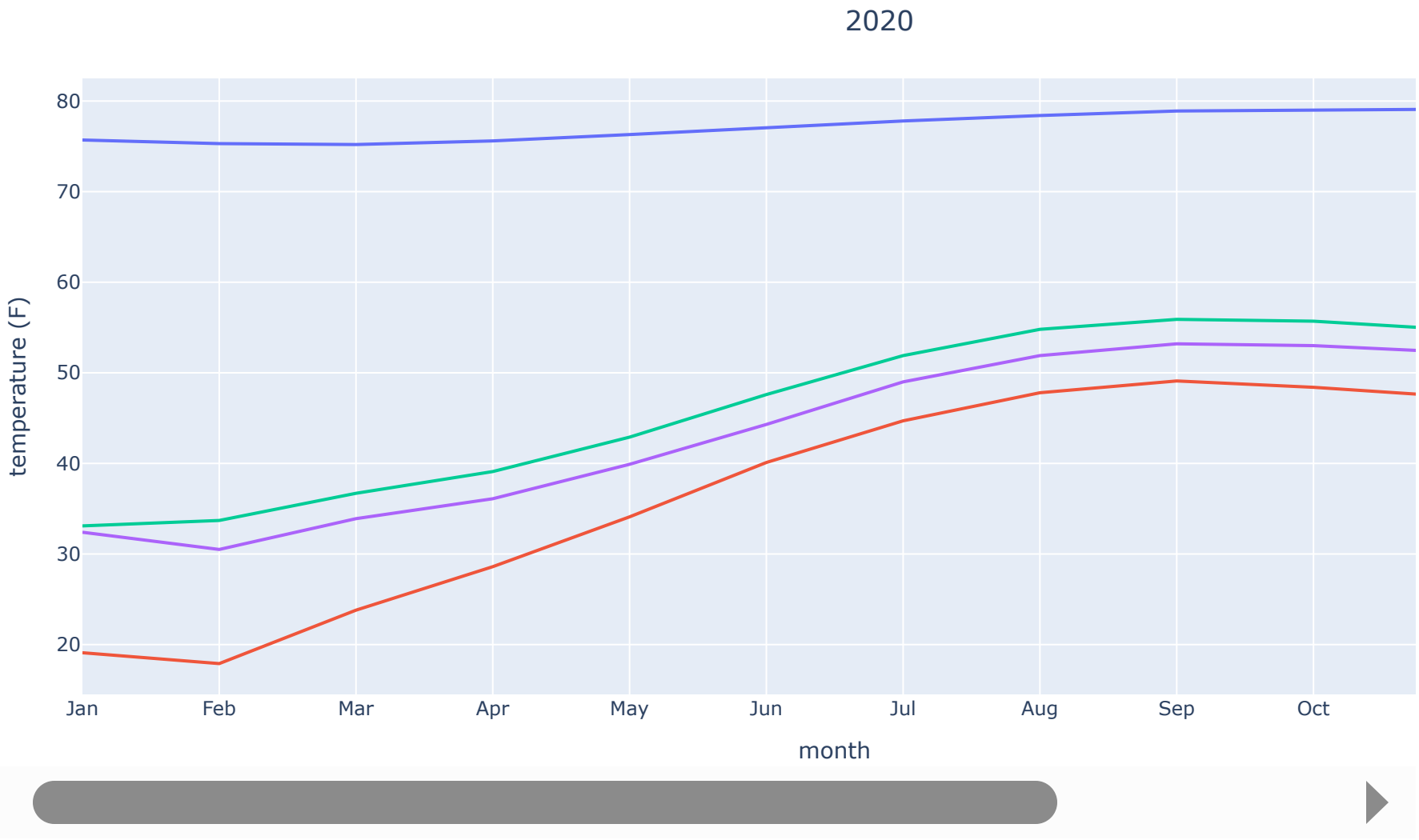
```
cities = list(df.columns[2:])
dfu = pd.melt(df,  id_vars=['Year', 'Month'], value_vars=cities, var_name='city', value_name='temp')
dfu.head()
```

	Year	Month	city	temp
0	1950	Jan	Honolulu	72.7
1	1950	Feb	Honolulu	73.7
2	1950	Mar	Honolulu	73.8
3	1950	Apr	Honolulu	73.9
4	1950	May	Honolulu	74.4

Exercise 2

Let's make a line graph of months by temperature for a given year across all cities. Since months are ordinal, it makes sense to connect successive datapoints with lines.

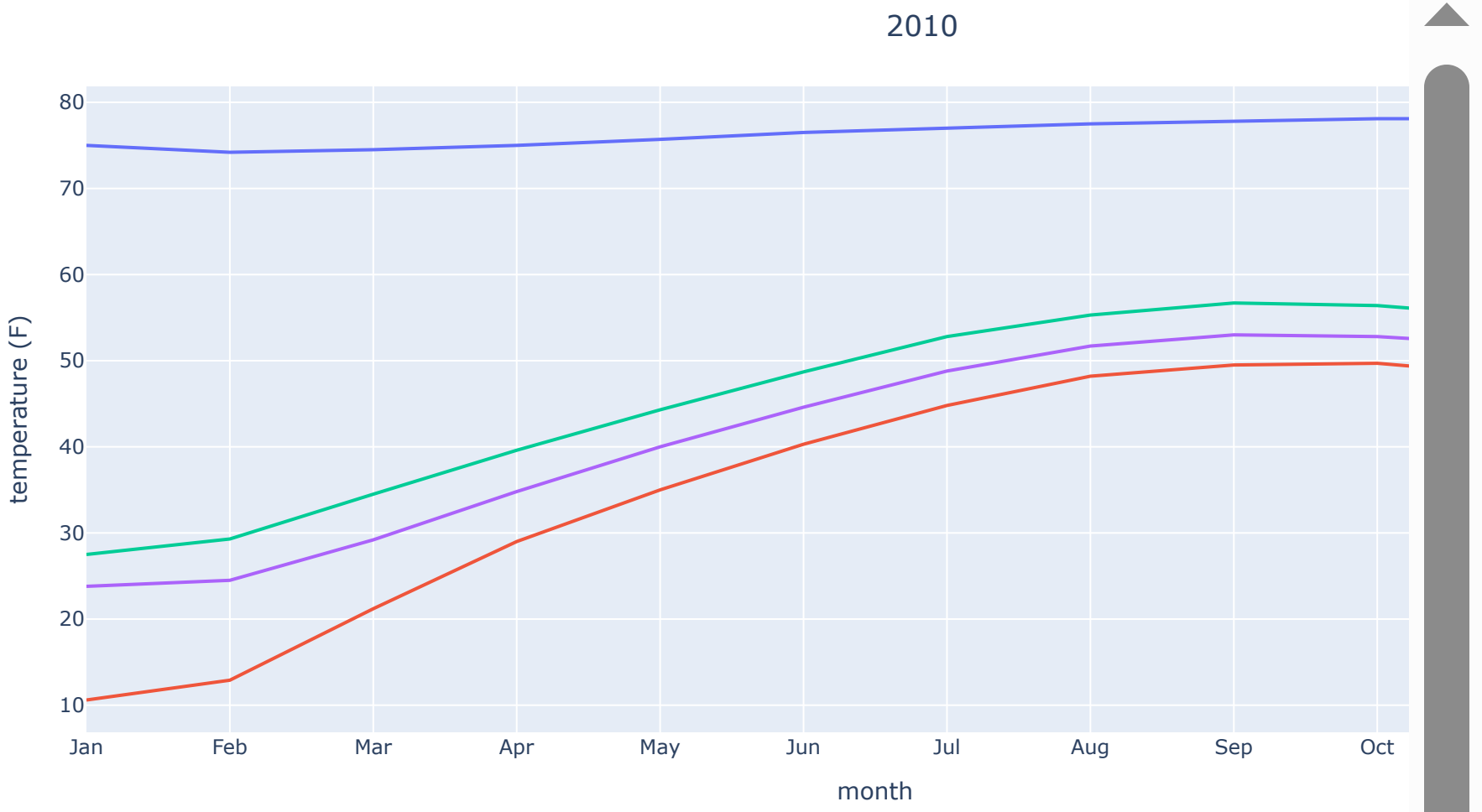
```
cities = list(df.columns[2:])
months = calendar.month_abbr[1:13]
year = 2020
# extract subframe of interest
df2 = df[df.Year == year]
fig = px.line(df2, x=months, y=cities)
fig.update_layout(title_text=year, title_x=0.5, legend_title_text='cities')
fig.update_layout(xaxis_title_text='month', yaxis_title_text='temperature (F)')
fig.show(renderer=renderer)
```

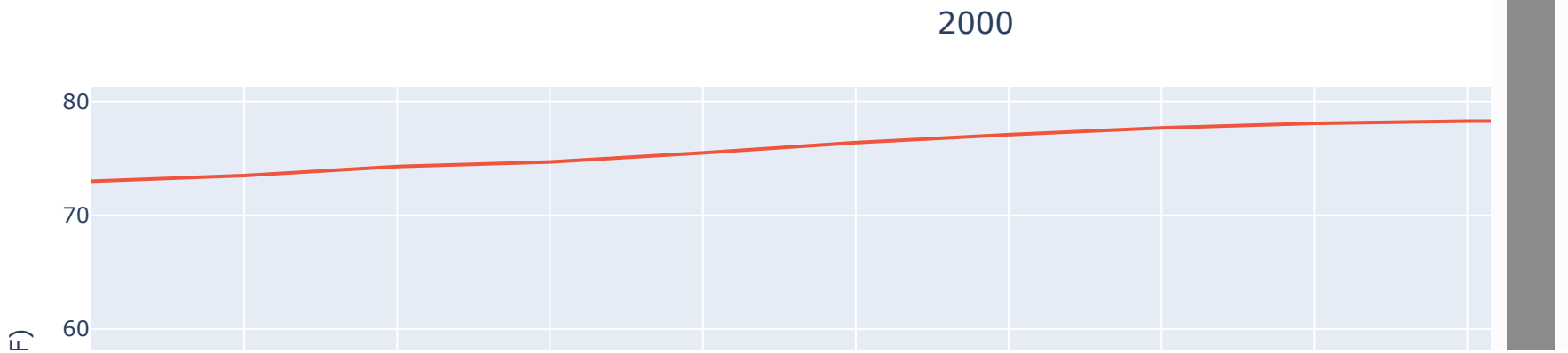


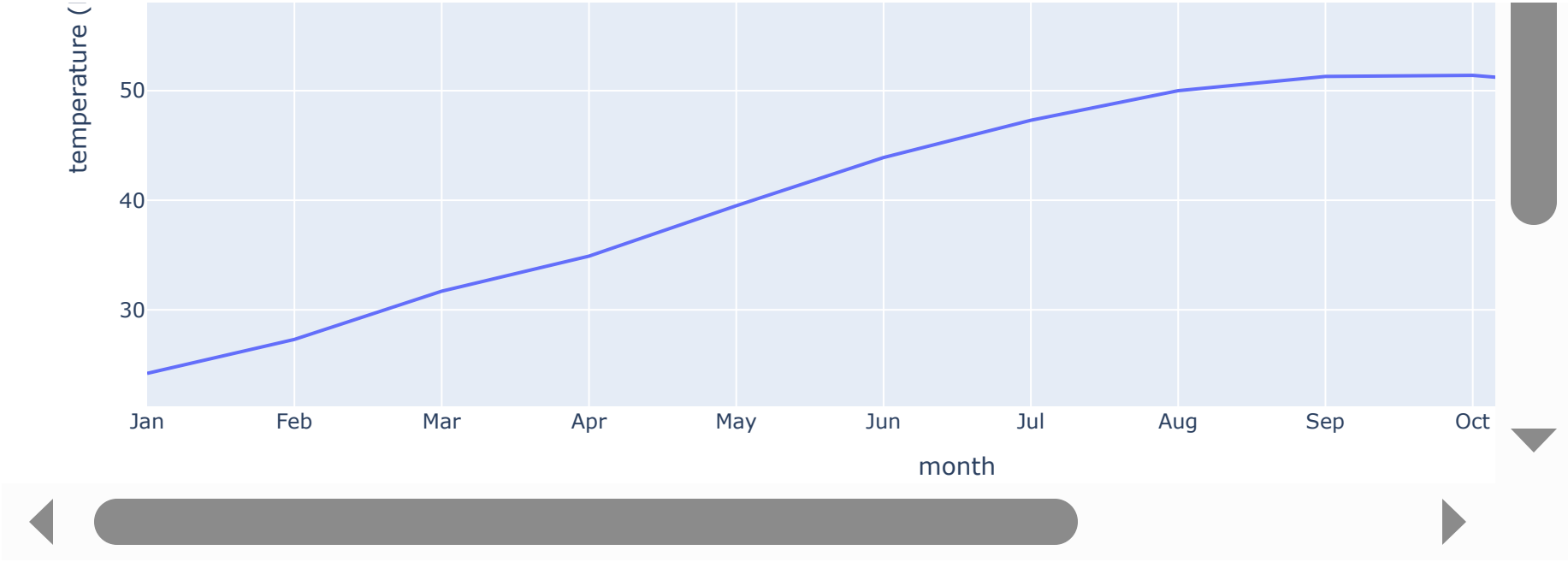
Double-click (or enter) to edit

```
# we bundle the previous code chunk as a function
def plot_one_year(df, year, cities=None):
  if cities:
    df = df[['Year', 'Month'] + cities]
  else:
    cities = list(df.columns[2:])
```

```
    months = calendar.month_abbr[1:13]
    df2 = df[df.Year == year]
    fig = px.line(df2, x=months, y=cities)
    fig.update_layout(title_text=year, title_x=0.5, legend_title_text='cities')
    fig.update_layout(xaxis_title_text='month', yaxis_title_text='temperature (F)')
    return fig

plot_one_year(df, 2010).show(renderer=renderer)
plot_one_year(df, 2000, ['Buffalo', 'Honolulu']).show(renderer=renderer)
```


2010
80
70
60
50
40
30
20
10
temperature (F)
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
month


2000
80
70
60



For this exercise, define the function

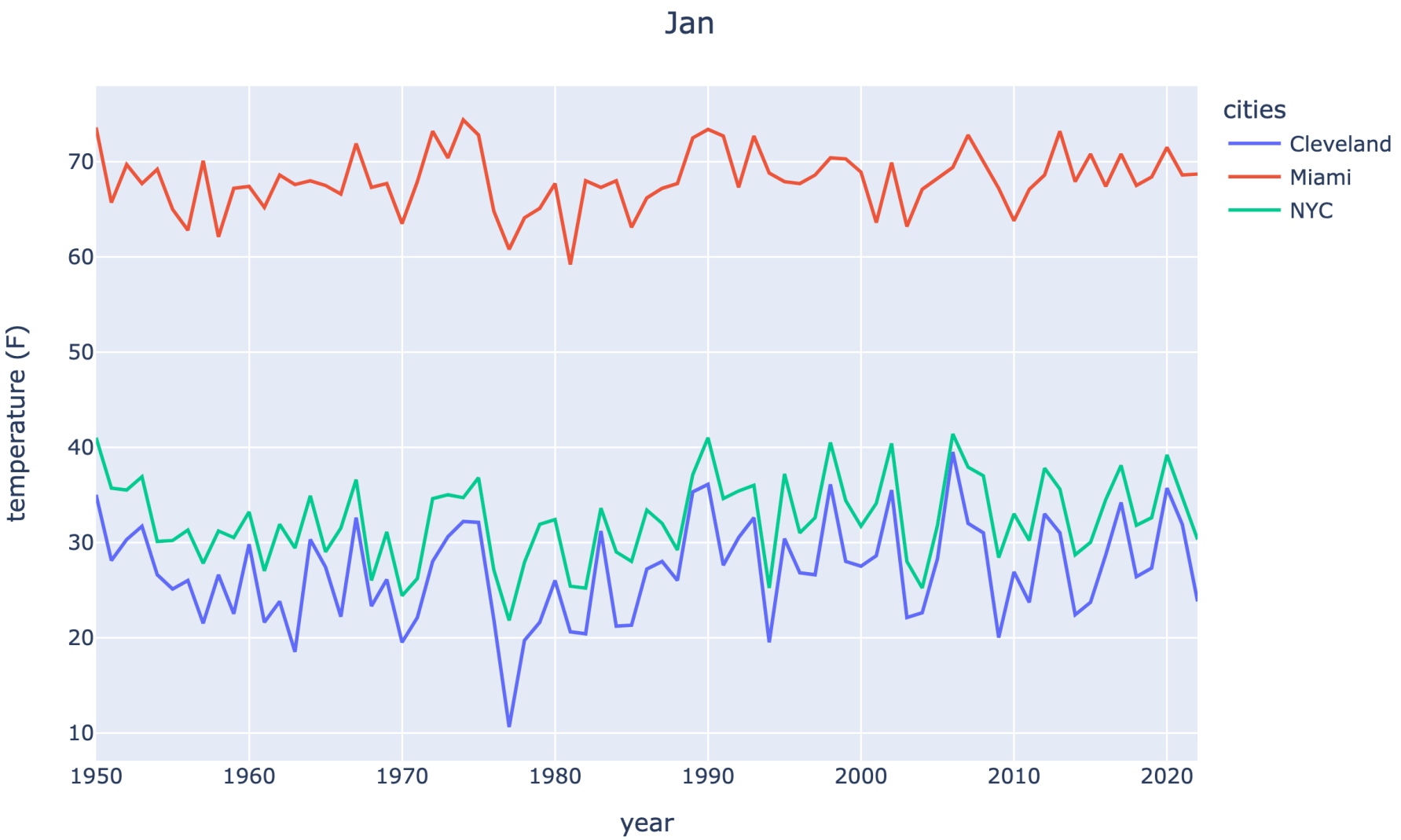
```
plot_one_month(frame, month, cities)
```

that draws a line graph over all years for given *month* and list of *cities*. Assume *frame* has one of the two formats (`df` or `df2`) defined above, though for this exercise you might find it easier to assume the `df` format.

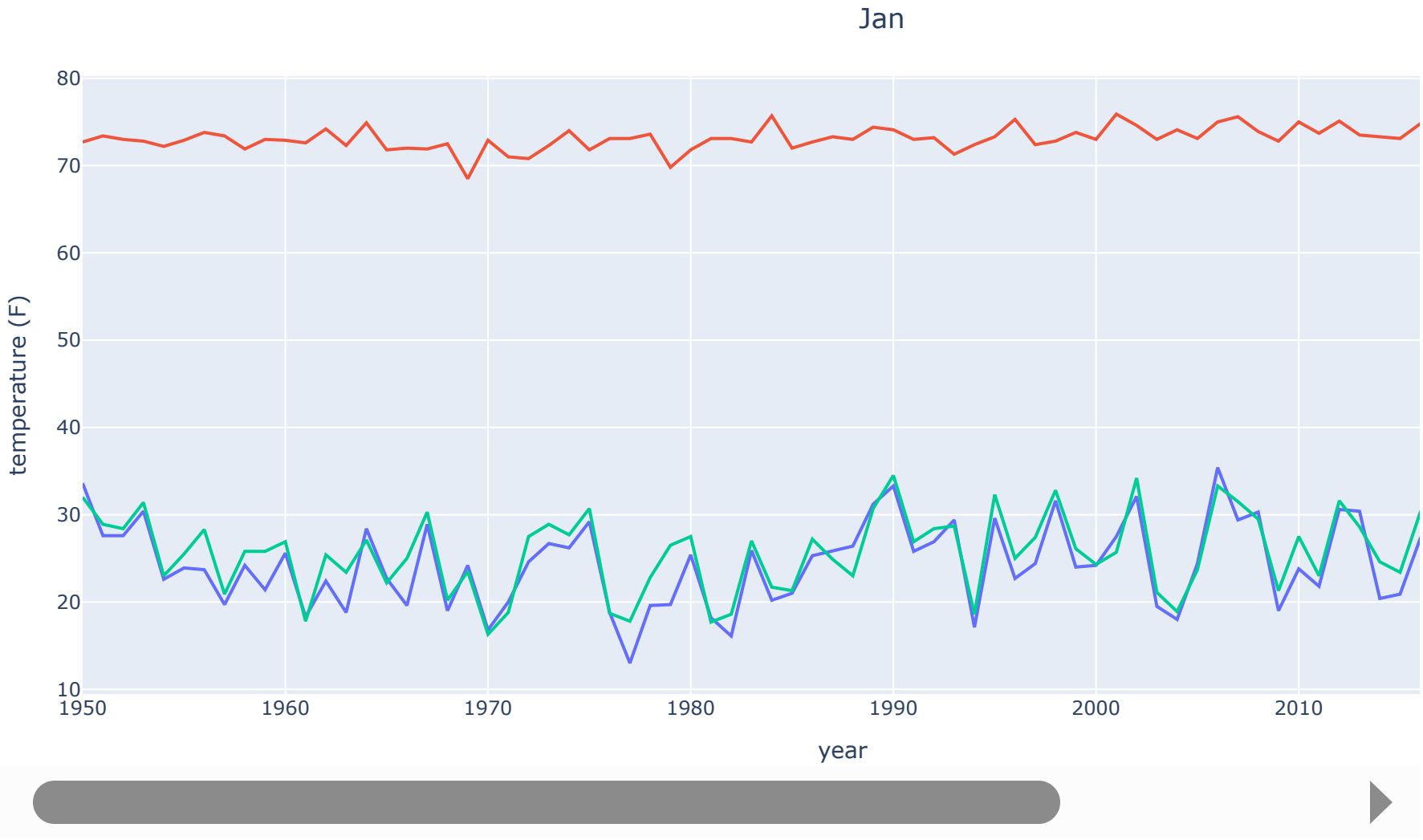
The following image results from the function call:

```
plot_one_month(frame, 'Jan', ['Cleveland', 'Miami', 'NYC']).show()
```

```
display_images(['ass2_ex2.png'])
```


Jan
cities
Cleveland
Miami
NYC
temperature (F)
70
60
50
40
30
20
10
1950
1960
1970
1980
1990
2000
2010
2020
year

```
def plot_one_month(frame, month, cities=None):
    #define cities
    if cities:
        frame = frame[['Year', 'Month'] + cities]
    else:
        cities = list(frame.columns[2:])
    #get list of unique years in dataset
    years = list(frame['Year'].unique())
    df2 = frame[frame.Month == month]
    fig = px.line(df2, x=years, y=cities)
    fig.update_layout(title_text=month, title_x=0.5, legend_title_text='cities')
    fig.update_layout(xaxis_title_text='year', yaxis_title_text='temperature (F)')
    return fig


plot_one_month(df, 'Jan', ['Buffalo', 'Honolulu', 'Hartford']).show()
```

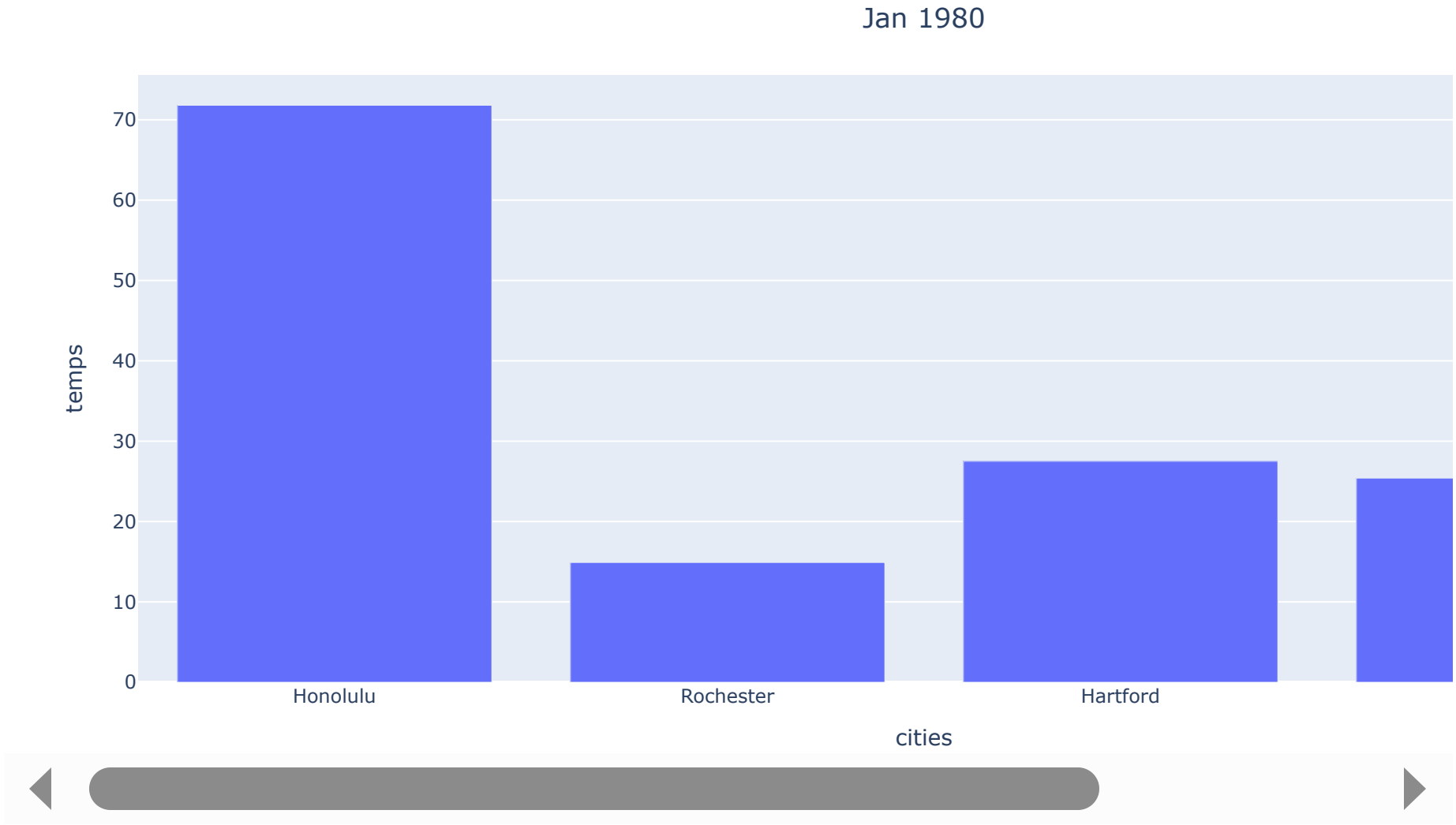


Exercise 3

We can use a bar graph to show the temperature for a given month and year for each city.

```
def barplot_year_month(df, year, month):
  cities = list(df.columns[2:])
  rcd = df[(df.Year == year) & (df.Month == month)][cities]
```

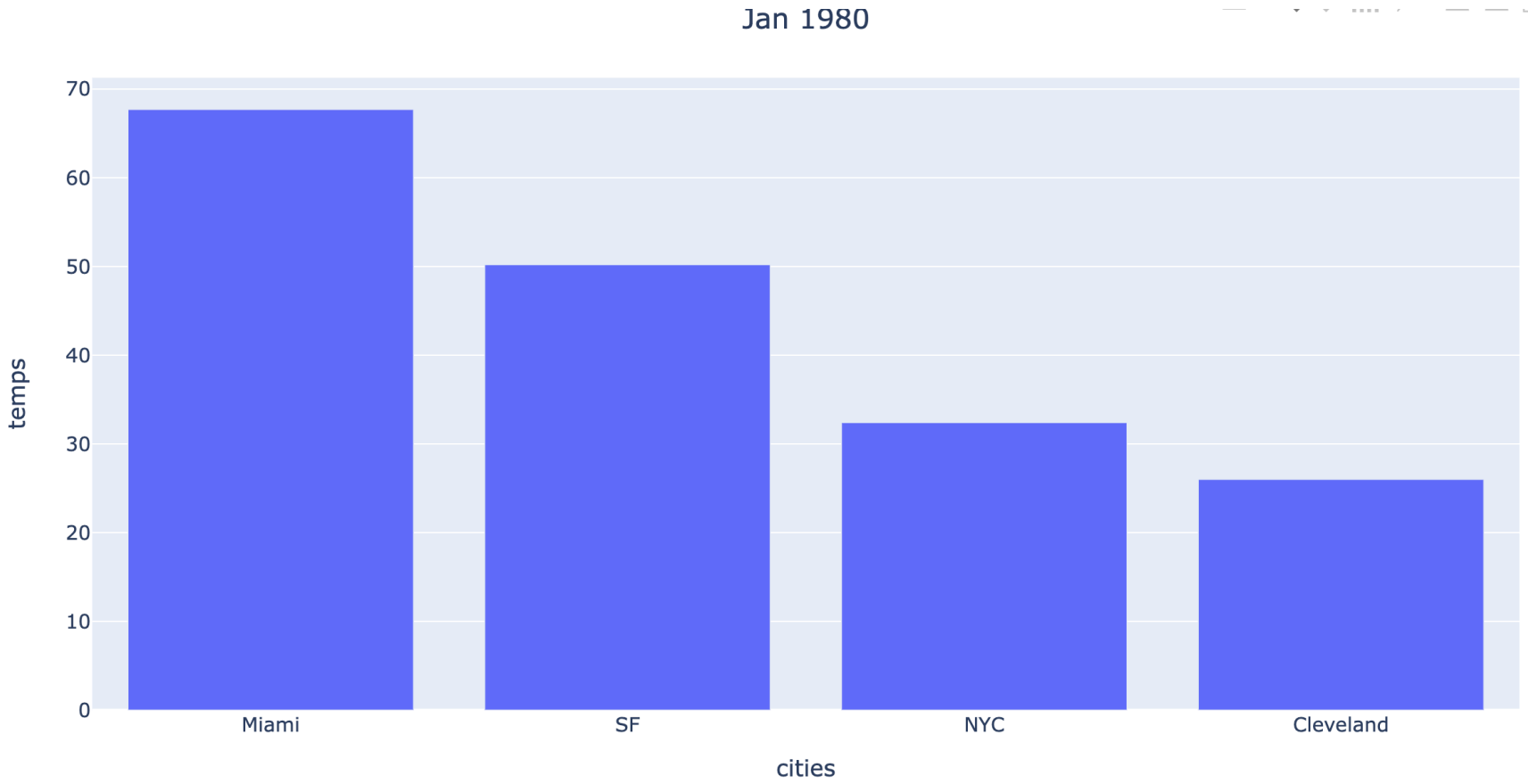
```
    df = pd.DataFrame({'cities': cities, 'temps': rcd.values[0]})
    fig = px.bar(df, x='cities', y='temps')
    title = str(month) + ' ' + str(year)
    fig.update_layout(title_text=title, title_x=0.5, legend_title_text='cities')
    return fig

barplot_year_month(df, 1980, 'Jan')
```



For this excercise, write a version of function `barplot_year_month` that generates a bar graph whose bars are ordered by descending height (from warmest to coldest).

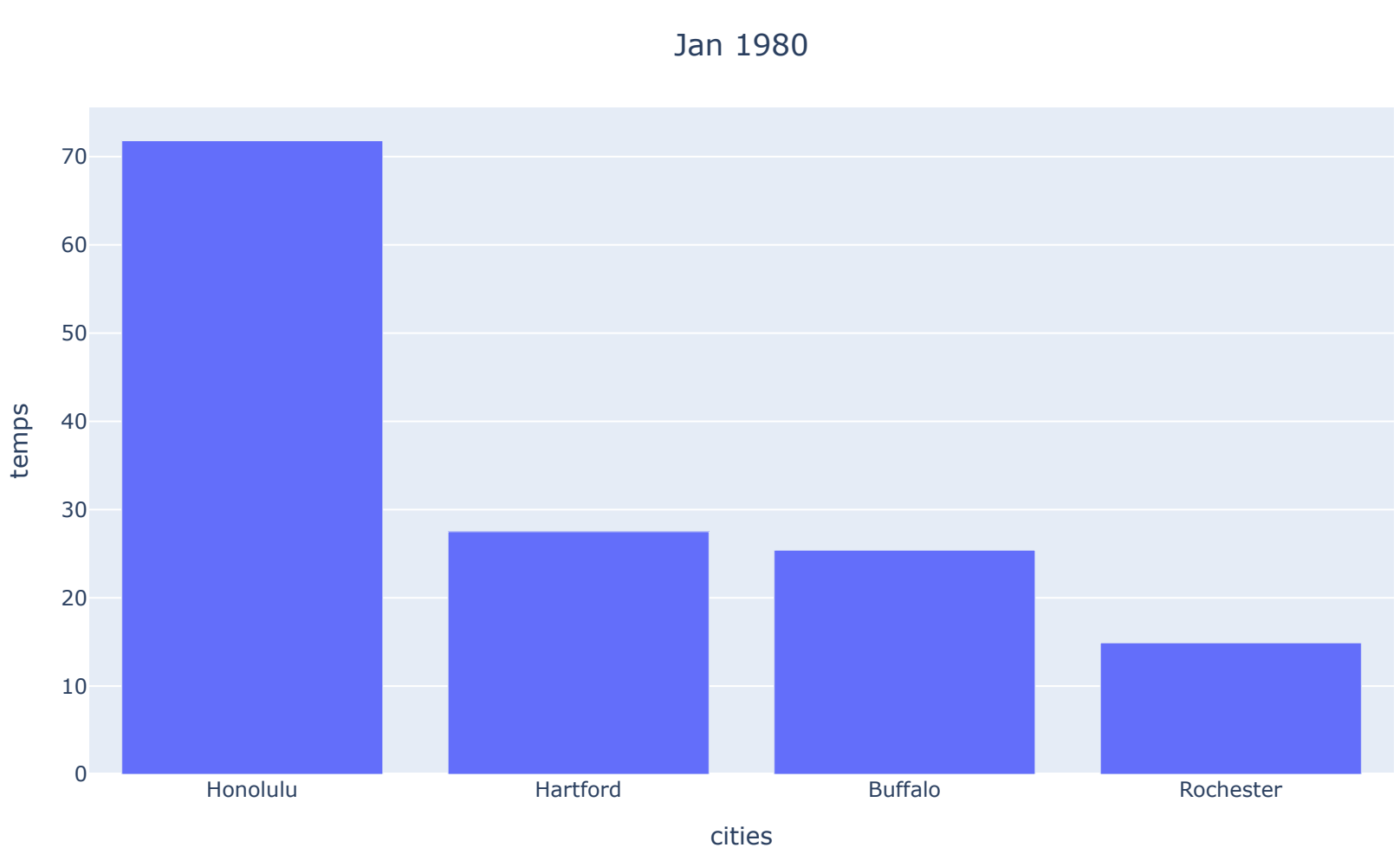
```
barplot_year_month(frame, 1980, 'Jan')
```

```
display_images(['ass2_ex3.png'])
```




```
def barplot_year_month(df, year, month):
  cities = list(df.columns[2:])
  rcd = df[(df.Year == year) & (df.Month == month)][cities]
  df = pd.DataFrame({'cities': cities, 'temps': rcd.values[0]})
  #sort the dataset in descending order
  df_sorted = df.sort_values(by='temps', ascending=False)
  fig = px.bar(df_sorted, x='cities', y='temps')
```

```
    title = str(month) + ' ' + str(year)
    fig.update_layout(title_text=title, title_x=0.5, legend_title_text='cities')
    return fig

barplot_year_month(df, 1980, 'Jan')
```



Exercise 4

Since the cities are unordered categorical data, we can show their temperatures using a dot plot. Each city's temperature values for a given year are depicted by a column of dots arranged along the vertical temperature axis. Write the function `dotplot_one_year(frame, year)` that returns the figure for such a plot.

For example, the following plot is produced by this call:

```
dotplot_one_year(frame, 2012)
```

Hint: For this exercise, you might find it easier to work with `df2` and write `dotplot_one_year2(frame, year)`, although either dataframe format can be used.

```
display_images(['ass2_ex4.png'])
```


```
def dotplot_one_year2(frame, year):
    df2 = frame[frame.Year == year]
    fig = px.scatter(df2, x='city', y='temp', color='city', opacity=0.6)
    fig.update_traces(marker_size=20)
    fig.update_layout(title_text=year,title_x=0.5, showlegend=False)
    fig.update_layout(xaxis_title_text='cities', yaxis_title_text='temperature (F)')
    return fig
```

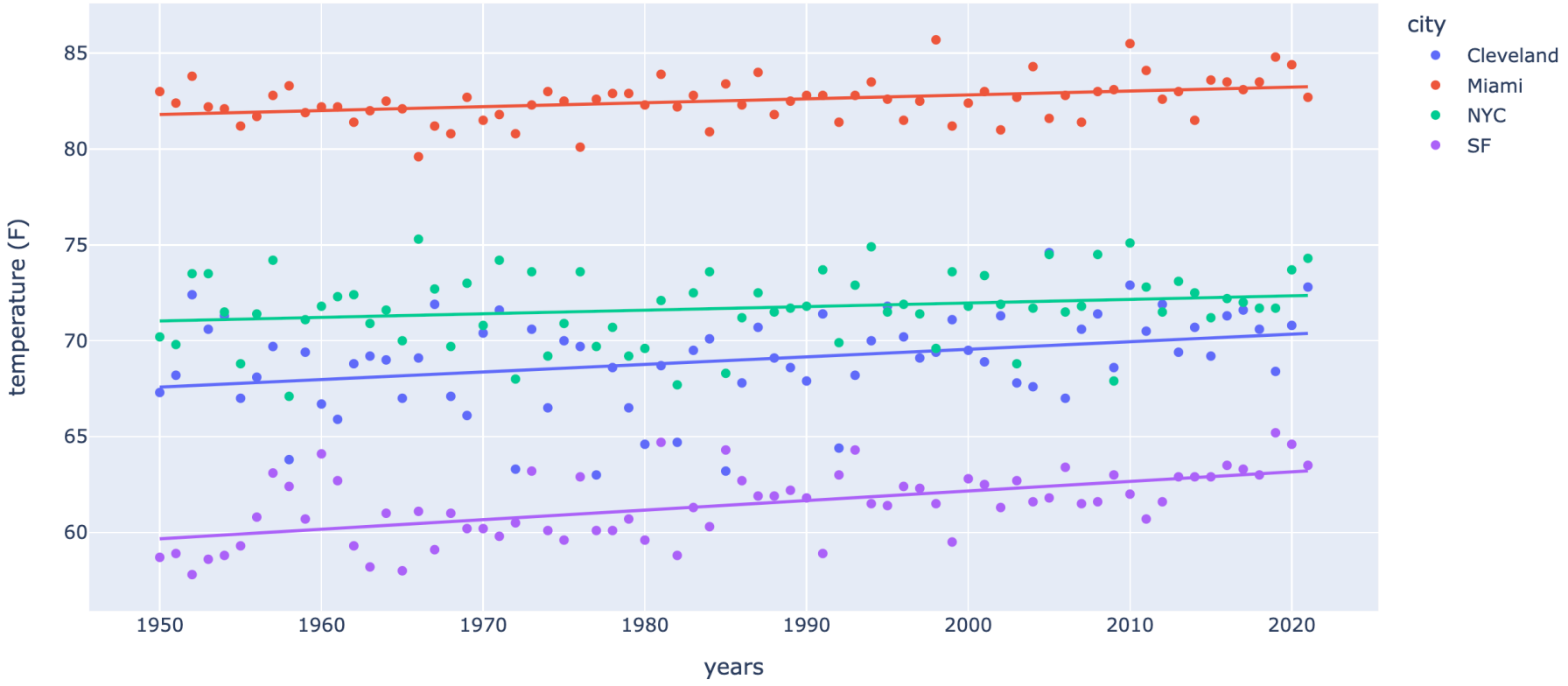
```
dotplot_one_year2(dfu, 2012)
```

Exercise 5

We can also use a scatterplot to show variation in temperature in a given month over the range of years, by city. Write the function `scatterplot_one_month(df, month)` that generates a scatterplot for each city on the same plot, which includes a best-fit line for each city.

Hint: Work with the `df2` dataframe format. Name your function `scatterplot_one_month2` to indicate this choice of format.

```
display_images(['ass2_ex5.png'])
```

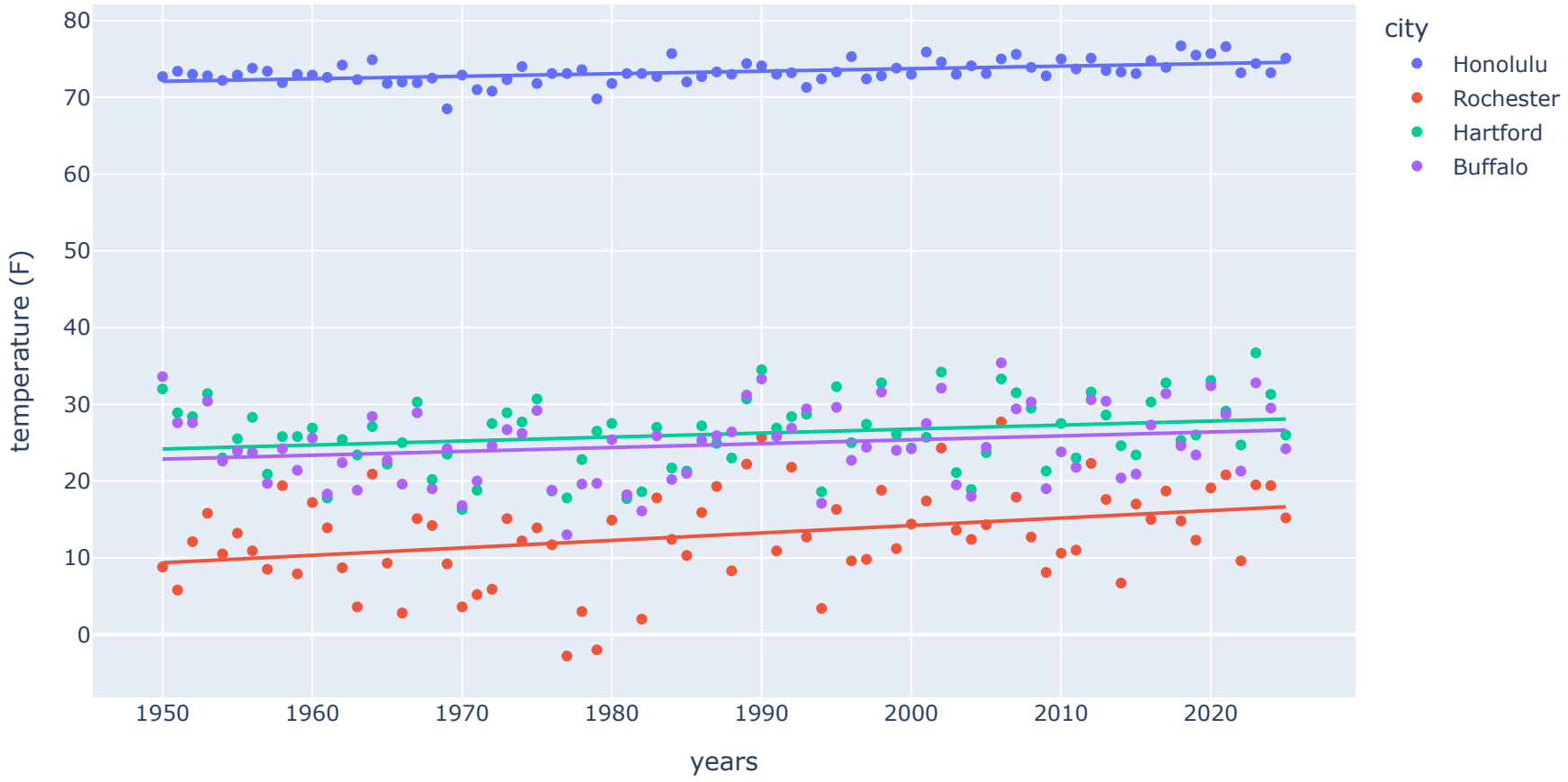



```
!pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.11/dist-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.0.2)
```

```
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.15.3
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.2.2
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->stats
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->sta
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas
```

```
def scatterplot_one_month2(df, month):
    cities = list(df.columns[2:])
    df2 = df[df.Month == month]
    fig = fig = px.scatter(df2, x="Year", y="temp", color="city",trendline='ols' )
    fig.update_layout(legend_title_text='city')
    fig.update_layout(xaxis_title_text='years', yaxis_title_text='temperature (F)')
    return fig

scatterplot_one_month2(dfu, 'Jan')
```



Exercise 6

We're given a function that takes *n* and returns a list *is_prime* of boolean values indicating the integers through n that are prime: *is_prime[i]* is true iff *i* is prime.

```
def is_primes_upto(n):
    is_prime = np.ones((n+1,), dtype=bool)
```

```
    is_prime[:2] = False
    lim = int(np.sqrt(len(is_prime))) + 1
    for i in range(2, lim):
        is_prime[2*i::i] = False
    return is_prime

for i, v in enumerate(is_primes_upto(12)):
  print(i, v)
```

```
0 False
1 False
2 True
3 True
4 False
5 True
6 False
7 True
8 False
9 False
10 False
11 True
12 False
```

Define a function `plot_primes(m, n)` that plots a matrix heatmap of an $m \times n$ array P where entry $P[i][j]$ has one color if the integer $p = n * i + j$ is prime, a different color if p is composite. Two examples:

```
plot_primes(10, 10)
plot_primes(20, 20, color_sequence=px.colors.qualitative.Dark24)
```

In the first example, yellow indicates prime and blue composite. In the first row, yellow squares correspond to the primes 2, 3, 5, and 7; in the second row, to the primes 11, 13, 17, and 19.

```
display_images(['ass2_ex6a.png', 'ass2_ex6b.png'])
```